Recurrent Neural Network Transducers (RNN-T)

Mark Hasegawa-Johnson

September 11, 2023

Outline

The Sequence-to-Sequence Problem

- Input sequence:

$$\mathbf{x} = (x_1, \ldots, x_T), \quad x_t \in \mathcal{X}$$

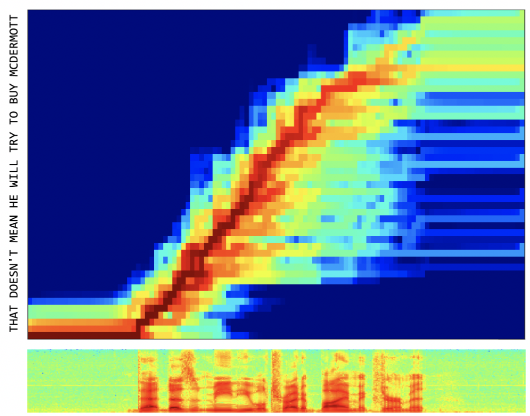
- Output sequence:

$$\mathbf{y} = (y_1, \ldots, y_U), \quad x_u \in \mathcal{Y}$$

- Goal: Learn a function that models $\Pr(\mathbf{y}|\mathbf{x})$

Example: Speech Recognition

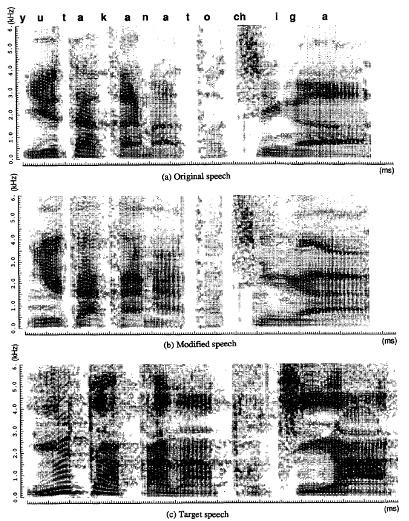
- Input: x_t = log spectral magnitude of frame t
- Output: $y_u \in \{1, \ldots, K\}$ is one of K symbols in an output alphabet, $\hat{y}_u \in \{0, 1\}^K$ is the corresponding one-hot vector.



Graves, 2012, `https://arxiv.org/abs/1211.3711`

Example: Voice Conversion

- Input: x_t = log spectral magnitude of input speech, frame t
- Output: y_u = log spectral magnitude of target speech, frame u

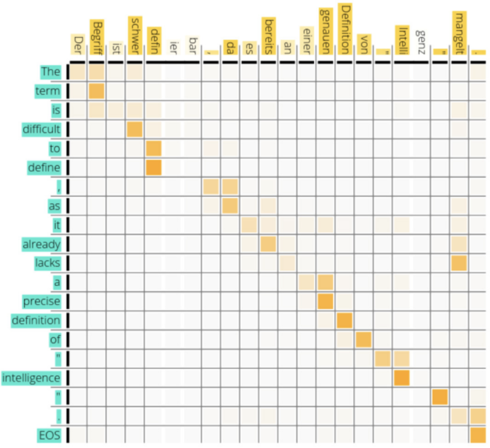


(a) Original speech
(b) Modified speech
(c) Target speech

Mizuno & Abe, 1995, https://doi.org/10.1016/

Example: Machine Translation

- Input: x_t = word selected from input sentence,
- Output: y_u = one-hot vector of output word



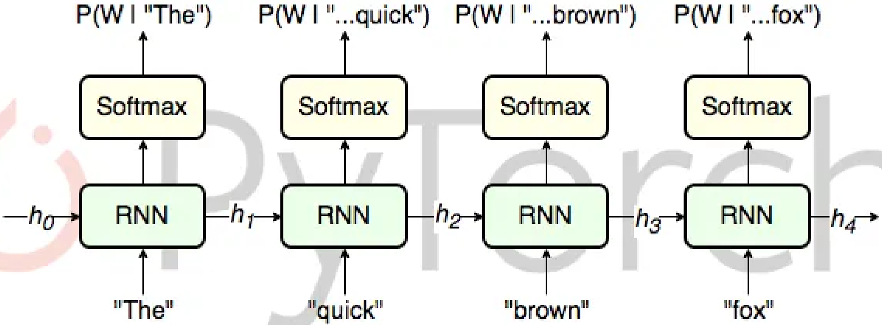
Munz et al., 2021, `https://doi.org/10.1016/j.cag.2021.12.003`

Outline

A Sequence-to-Sequence Network is made of two component networks

- Prediction Network: predict next output label given previous output labels (language model)
- Transcription Network: predict output label given inputs (acoustic model)

Prediction Network/Language Model



Florijan Stamenković, medium.com

Prediction Network/Language Model

In a language model, g_u is the logit output vector, and $\Pr(y_{u+1}=k|y_1,\ldots,y_u)=\text{softmax}_k(g_u)$. The computation of g_u is autoregressive in different ways, depending on the decoder architecture:

- RNN:

$$h_u = \mathcal{H}\left(W_{ih}\hat{y}_u + W_{hh}h_{u-1} + b_g\right)$$
$$g_u = W_{ho}h_u + b_o$$

- Autoregressive Transformer: $\hat{\mathbf{y}}_{[1:u]} = [\hat{y}_1,\ldots,\hat{y}_u]$, and

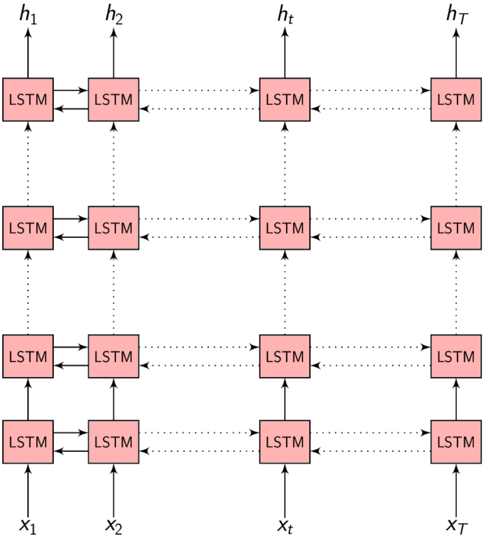
$$\alpha_{u,u'} = \text{softmax}_{u'}\left(g_{u-1}^T W_Q^T W_K \hat{\mathbf{y}}_{[1:u']}\right)$$
$$g_u = \sum_{u'} \alpha_{u,u'} W_V \hat{y}_{u'}$$

Transcription Network

The transcription network converts an input sequence, $\mathbf{x} = (x_1, \ldots, x_T)$, into a sequence of vector embeddings, $\mathbf{f} = (f_1, \ldots, f_T)$. There are many well-studied ways to do this, e.g.,

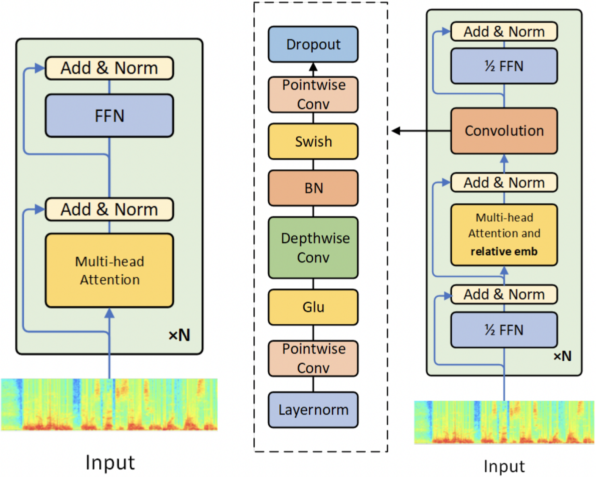
- Bidirectional LSTM
- Transformer
- Conformer

BLSTM Encoder



Shree, `https://vak.ai/speech/attention/encoder-decoder-basics/`

Transformer and Conformer



(a) Transformer (b) Conformer

Li, 2021, `https://arxiv.org/abs/2111.01690`

Sequence-to-Sequence: Component Networks

- The prediction network computes g_u from $\mathbf{y}_{[1:u]} = (y_1, \ldots, y_u)$
- The transcription network computes f_t from $\mathbf{x} = (x_1, \ldots, x_t, \ldots, x_T)$

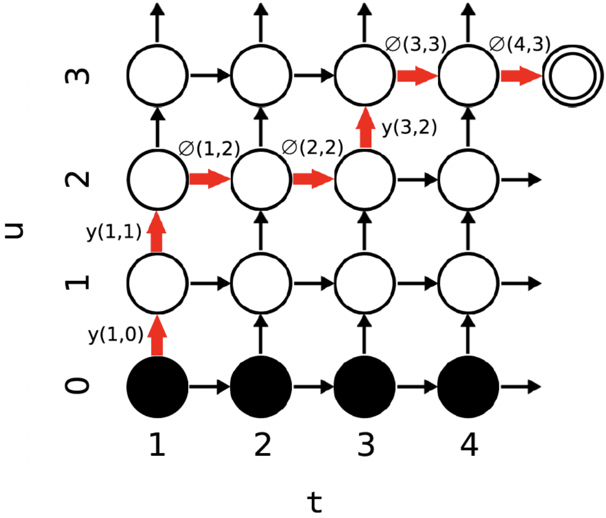
Outline

Alignment Sequence

The RNN-T aligns input sequence $\mathbf{x}$ to output sequence $\mathbf{y}$ by using an alignment sequence, $\mathbf{a} = (a_{1,0}, \ldots, a_{T,U})$, where

- An **upward step**, $a_{t,u} = y_{u+1}$, means that input character x_t generates output character y_u.
- A **rightward step**, $a_{t,u} = \varnothing$, means that there are no more output characters generated by x_t, so we should move forward to start looking at x_{t+1}.

Upward Steps and Rightward Steps



Graves, 2012, `https://arxiv.org/abs/1211.3711`

Key Idea: Combine information from LM and AM

The key idea of RNNT is that the next character depends on both the language model and the acoustic model:

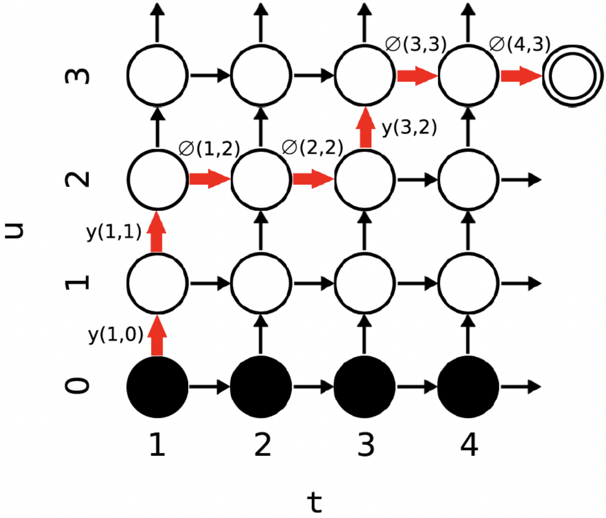
$$\Pr(a_{t,u}=k|\mathbf{y}_u,\mathbf{x}) = \text{softmax}(f_t^k+g_u^k) = \frac{\exp(f_t^k+g_u^k)}{\sum_{k'}\exp(f_t^{k'}+g_u^{k'})}$$

Alignment Probabilities

During training, the desired sequence $\mathbf{y} = (y_1, \ldots, y_U)$ is known, so we can define these useful shorthands:

$$y(t,u) \equiv \Pr(y_{u+1}|t,u)$$
$$\varnothing(t,u) \equiv \Pr(\varnothing|t,u)$$

Alignment Probabilities



Graves, 2012, `https://arxiv.org/abs/1211.3711`

Forward-Backward Probabilities

In order to train the neural network, we need these two probabilities:

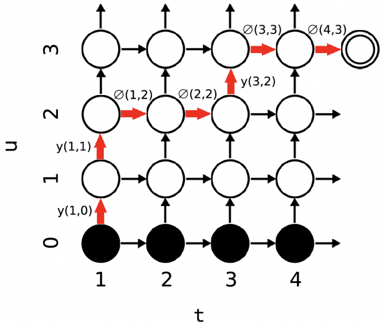
- Forward probability:

$$\begin{aligned}\alpha(t,u) &\equiv \Pr(\mathbf{y}_{[1:u]}|\mathbf{x}_{[1:t]})\\ &= \alpha(t-1,u)\varnothing(t-1,u) + \alpha(t,u-1)y(t,u-1)\end{aligned}$$

- Backward probability:

$$\begin{aligned}\beta(t,u) &\equiv \Pr(\mathbf{y}_{[(u+1):U]}|\mathbf{x}_{[t:T]})\\ &= \beta(t+1,u)\varnothing(t,u) + \beta(t,u+1)y(t,u)\end{aligned}$$

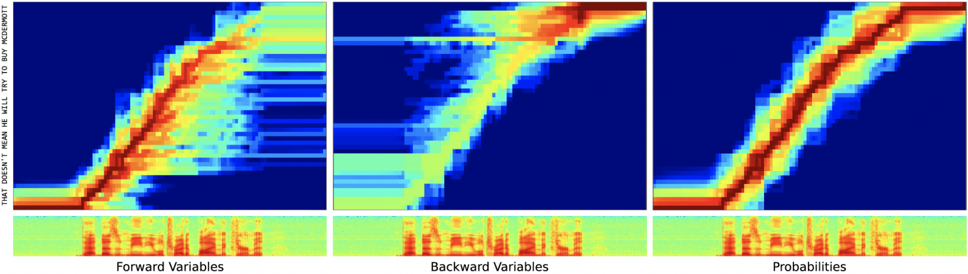
Total Alignment Probability



The probability that **x** generates **y**, and that y_u is generated no later than x_t, is

$$\Pr(\mathbf{y}_{[1:u]}|\mathbf{x}_{[1:t]})\Pr(\mathbf{y}_{[(u+1):U]}|\mathbf{x}_{[t:T]}) = \alpha(t,u)\beta(t,u)$$

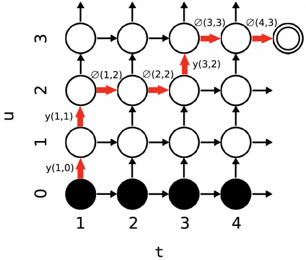
Total Alignment Probability



Graves, 2012, `https://arxiv.org/abs/1211.3711`

Total Sequence Probability

Notice that, to get from cell $(1,0)$ to cell $(5,3)$, the network must pass through **one and only one** of the cells on any descending diagonal. For example, it must pass through **one and only one** of the cells $\{(1,3),(2,2),(3,1),(4,0)\}$.



The total probability of $\mathbf{y}$ given $\mathbf{x}$ is therefore:

$$\Pr(\mathbf{y}|\mathbf{x}) = \sum_{(t,u):t+u=n} \alpha(t,u)\beta(t,u)$$

Outline

Training Loss

The training loss is negative log-probability:

$$\mathcal{L} = -\ln \Pr(\mathbf{y}|\mathbf{x})$$

As usual, the derivative of log-probability has a nice normalizing factor:

$$\frac{\partial \mathcal{L}}{\partial \beta(t,u)} = -\frac{1}{\Pr(\mathbf{y}|\mathbf{x})} \frac{\partial \Pr(\mathbf{y}|\mathbf{x})}{\partial \beta(t,u)}$$

Since $\Pr(\mathbf{y}|\mathbf{x}) = \sum_{(t,u):t+u=n} \alpha(t,u)\beta(t,u)$,

$$\frac{\partial \mathcal{L}}{\partial \beta(t,u)} = -\frac{\alpha(t,u)}{\Pr(\mathbf{y}|\mathbf{x})}$$

Loss Gradient

Remember that the backward probability is

$$\begin{aligned}\beta(t,u) &= \beta(t+1,u)\varnothing(t,u) + \beta(t,u+1)y(t,u)\\ &= \beta(t+1,u)\Pr(a_{t,u}=\varnothing|t,u) + \beta(t,u+1)\Pr(a_{t,u}=y_{u+1}|t,u)\end{aligned}$$

The derivative of the loss w.r.t. the softmax outputs is therefore:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\Pr(a_{t,u}|t,u)} &= \frac{\partial\mathcal{L}}{\partial\beta(t,u)}\frac{\partial\beta(t,u)}{\partial\Pr(a_{t,u}|t,u)}\\ &= -\frac{\alpha(t,u)}{\Pr(\mathbf{y}|\mathbf{x})}\begin{cases}\beta(t,u+1) & \text{if } a_{t,u}=y_{u+1}\\ \beta(t+1,u) & \text{if } a_{t,u}=\varnothing\end{cases}\end{aligned}$$

From there, we can use the usual gradient of the softmax to backpropagate.

Outline

RNN-T during test time

- During test time, we don't know **y**, so we need to consider every possible sequence.
- This could be a problem, because the number of such sequences is $|\mathcal{Y}|^U$.
- Graves uses a beam search for this. For each t, he keeps a finite buffer of the best partial paths.
- He takes advantage of the fact that the softmax output is less than or equal to one, $\Pr(a_{t,u}|t,u) \leq 1$, and therefore any extension of a path reduces its probability:

$$\Pr(y_1,\ldots,y_{u+1}|t,u+1) \leq \Pr(y_1,\ldots,y_u|t,u)$$
$$\Pr(y_1,\ldots,y_u|t+1,u) \leq \Pr(y_1,\ldots,y_u|t,u)$$

The RNN-T Beam Search

For t in 1 to T:

- Create an empty beam at time $t+1$
- While the beam at time $t+1$ contains fewer than W sequences that are more probable than the best sequence at time t:
  - Find the best sequence in the beam at time t, i.e., the one with the best $\Pr(y_1,\ldots,y_u|t,u)$. Remove it from the beam at time t, and instead, put its **rightward step**, $\Pr(y_1,\ldots,y_u|t+1,u)$, into the beam at time $t+1$.
  - Now compute all of the **upward steps**, $\Pr(y_1,\ldots,y_u,y_{u+1}|t,u+1)$ for all $y_{u+1}\in\mathcal{Y}$, and put them back into the beam at time t. If any of these is larger than the W^{th}-best at time $t+1$, then it will cause this loop to trigger again.

Outline

Hidden Markov Model

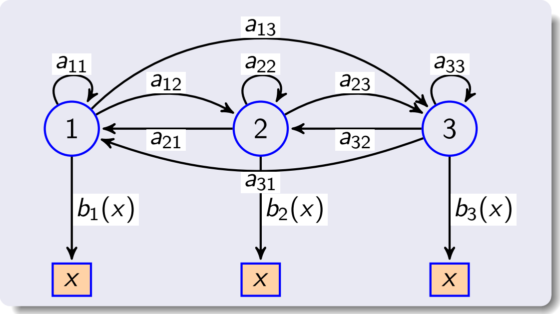
An HMM defines two probabilities:

- Transition probability:

$$a_{u,u+1} = \Pr(y_{u+1}|y_u)$$

- Likelihood:

$$b_u(x_t) = \Pr(x_t|y_u)$$



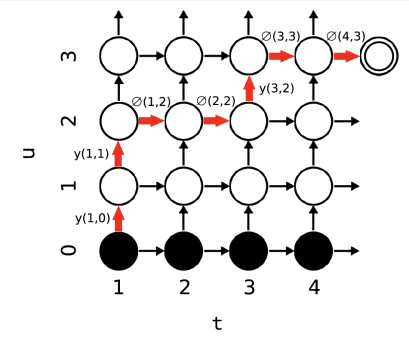
Hidden Markov Model

An HMM is a special case of an RNN-T in which the transitions depend only on local context, not global context:

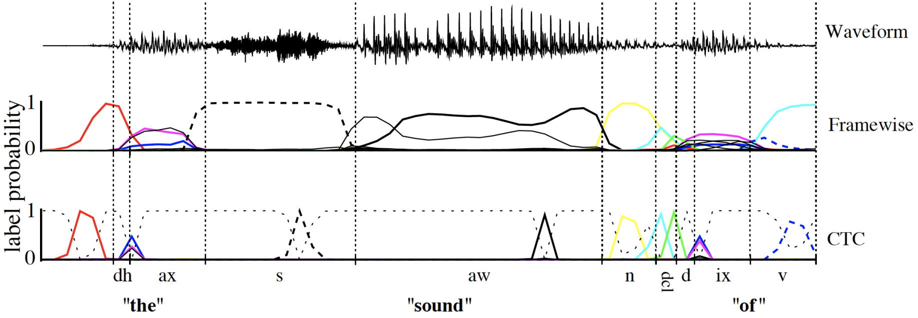
$$y(t,u) = a_{u,u+1}$$

$$\varnothing(t,u) = \begin{cases} a_{u,u}b_u(x_t) \\ \text{if prev. transit. was rightward,} \\ \varnothing(t,u) = b_u(x_t) \\ \text{if prev. transit. was upward} \end{cases}$$

RNN-T also renormalizes at each time step, but actually, so do most practical implementations of HMM.



Connectionist Temporal Classification (CTC)



CTC (Graves et al., 2006) is a special case of RNN-T in which there is no prediction model, only a transcription model. The alignment probabilities are therefore:

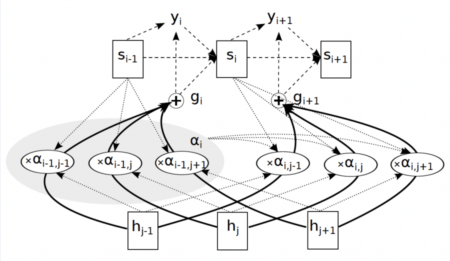
$$\Pr(a_{t,u} = k|\mathbf{x}) = \text{softmax}(f_t^k), \quad k \in \mathcal{Y} \cup \{\varnothing\}$$

Attention-Based Encoder-Decoder

An AED model generalizes RNN-T by permitting cross-attention to summarize the input, not just self-attention:

$$\Pr(a_{t,u} = k|\mathbf{x})$$
$$= \text{softmax}\left(s_u, \sum_t \alpha_{u,t} h_t\right)$$

This increases flexibility, but slows both training and testing, makes it more difficult to include a language model or other external information, and makes streaming harder.



Chorowski et al., 2015, https://proceedings.neurips.cc/paper/2015/hash/1068c6e4c8051cfd4e9ea8072e3189e html

Outline

Summary

- RNN-T generalizes both CTC and HMMs.
- Compared to HMM: it takes more data to train, but is more flexible.
- Compared to CTC: The only extra computational cost is the language model, which is usually low-cost compared to the acoustic encoder.
- RNN-T is most useful if you have an external source of information you'd like to use, e.g., a language model, parsing model, pronunciation model, dialog context model, or some other prediction model.
- This lecture was inspired by Desh Raj's blog post about RNN-T papers at Interspeech 2023: `https://desh2608.github.io/2023-08-28-interspeech-23-transducers/`.